M7-L2 Problem 1

In this function you will:

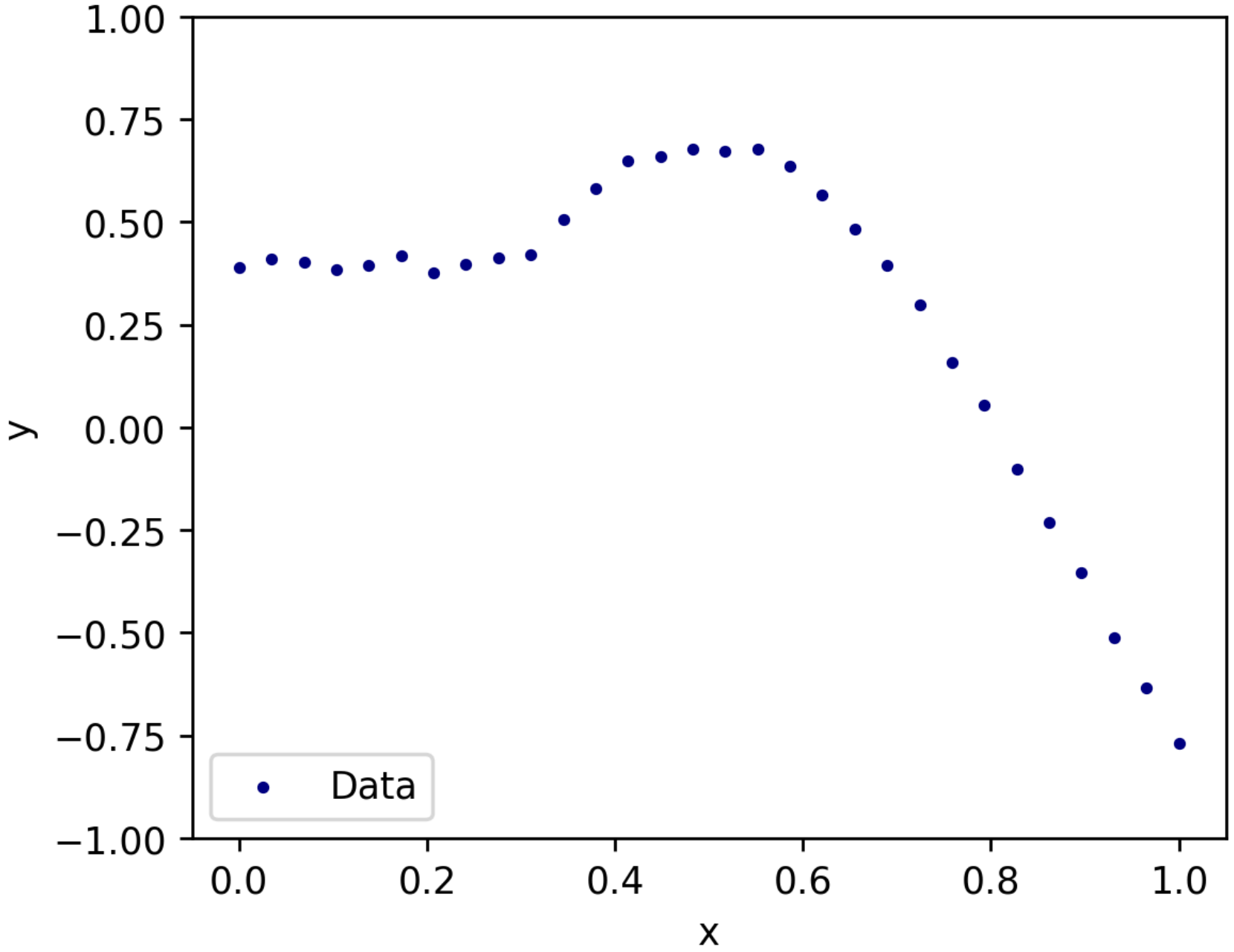
- Learn to use SciKit-Learn's `MLPRegressor()` model
- Look at the loss curve of an sklearn neural network
- Try out multiple activation functions

First, load the data in the following cell. This is the same data from M7-L1-P2

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPRegressor

x = np.array([0.        , 0.03448276, 0.06896552, 0.10344828, 0.13793103,0.17241379, 0.20689655, 0.24137931, 0.27586207
y = np.array([ 0.38914369,  0.40997345,  0.40282978,  0.38493705,  0.394214  ,0.41651437,  0.37573321,  0.39571087,  0.4


plt.figure(figsize=(5,4),dpi=250)
plt.scatter(x,y,s=5,c="navy",label="Data")
plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



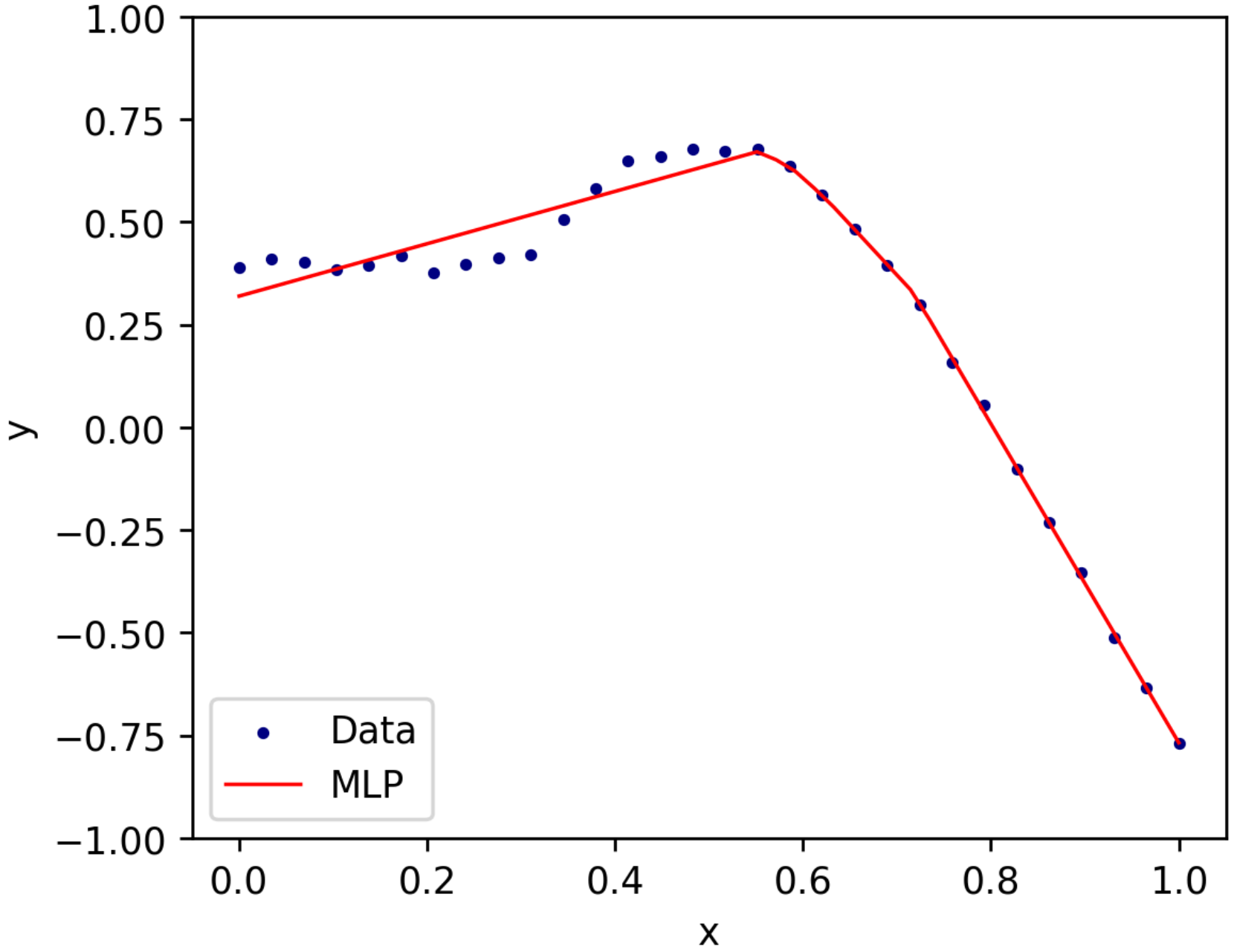

```
MLPRegressor()
```

Here, we create a simple MLP Regressor in sklearn and plot the results. The model is created and fitted in the same way as any other sklearn model. We choose hidden layer sizes 10,10. Note that our input and output are both 1-D, but we don't need to specify this at initialization.

In [2]:
```
mlp = MLPRegressor(hidden_layer_sizes=[10,10],max_iter = 10000, tol = 1e-10) # Tune here
mlp.fit(x, y)

xs = np.linspace(0,1)
ys = mlp.predict(xs.reshape(-1,1))

plt.figure(figsize=(5,4),dpi=250)
plt.scatter(x,y,s=5,c="navy",label="Data")
plt.plot(xs,ys,"r-",linewidth=1,label="MLP")
plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```



Tuning training hyperparameters

Chances are, the model above did a poor job fitting the data. Try changing the following parameters when initializing the `MLPRegressor` in the cell above:

- `max_iter` (this will need to be very large)
- `tol` (this will need to be very small)

You can read about what these do at https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html

Question

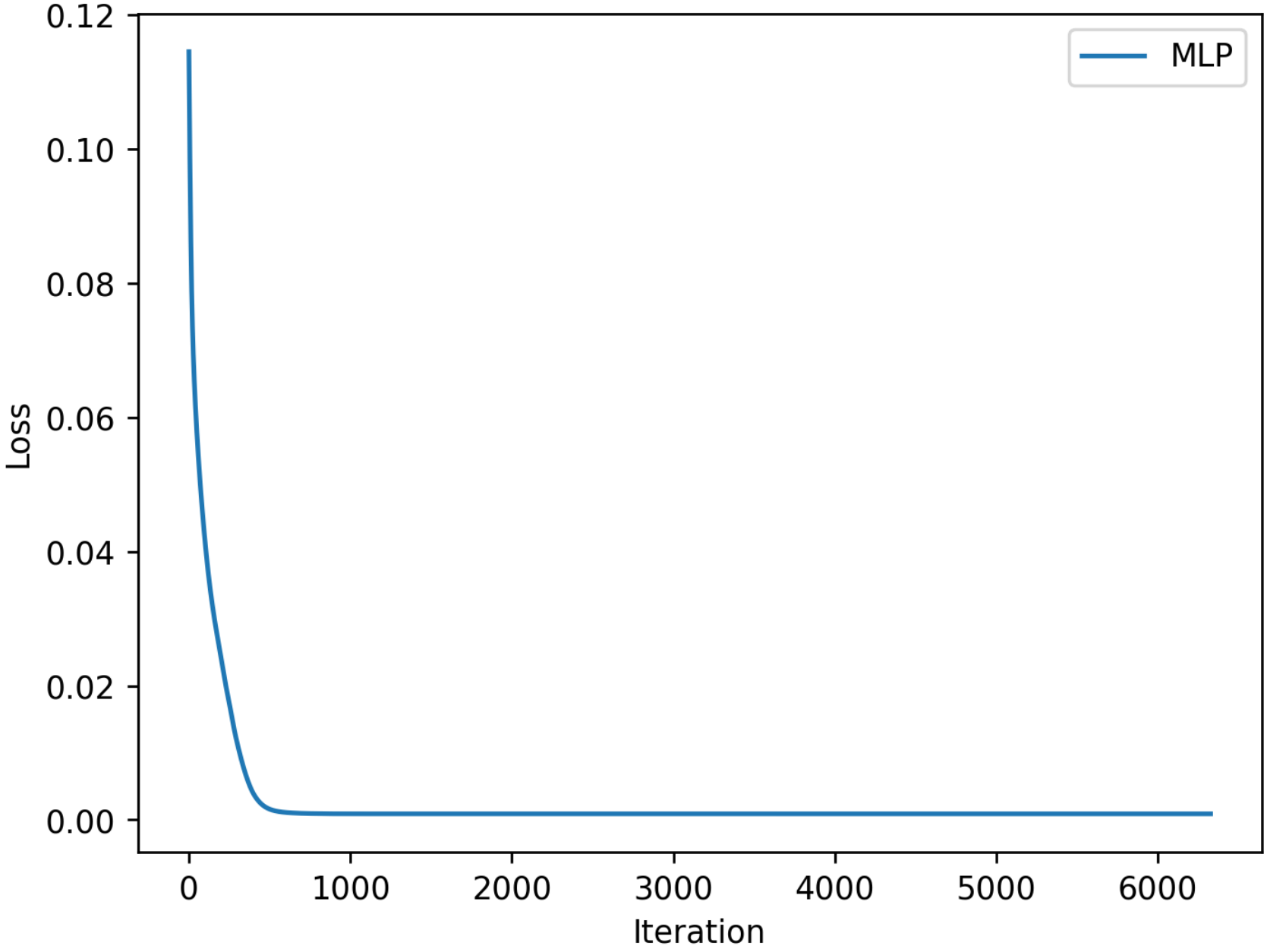
1. What values of `max_iter` and `tol` gave you a reasonable fit?

max_iter = 10000 and tol = 1e-10

Loss Curve

We can look at the loss curve by accessing `mlp.loss_curve_`. Let's plot this below:

In [3]:
```
loss = mlp.loss_curve_
plt.figure(dpi=250)
plt.plot(loss,label="MLP")
plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Activation Functions

Sklearn provides the following activation functions:

- `"identity"` (This is a linear function, it should not give good results)
- `"logistic"` (We call this 'sigmoid', although both this and tanh are sigmoid functions)
- `"tanh"`
- `"relu"`

Run the following cell to train a model on each. They can be accessed via, for example: `models["relu"]` for the relu activation model

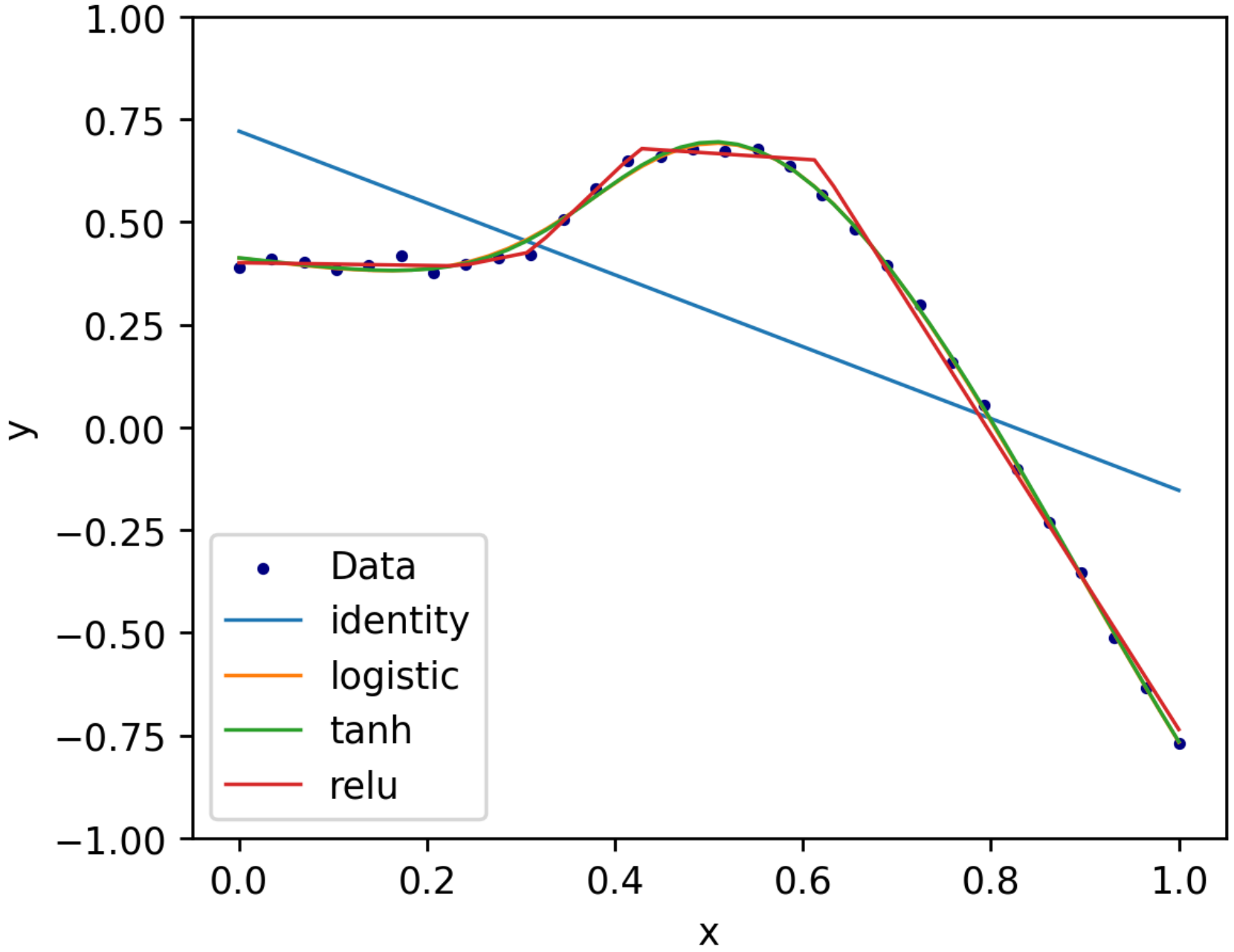
In [4]:
```
activations = ["identity","logistic","tanh","relu"]
models = dict()

for act in activations:
    model = MLPRegressor([10,10],random_state=50, activation=act,max_iter=100000,tol=1e-11)
    model.fit(x,y)
    models[act] = model

xs = np.linspace(0,1)
plt.figure(figsize=(5,4),dpi=250)
plt.scatter(x,y,s=5,c="navy",label="Data")

for act in activations:
    model = models[act]
    ys = model.predict(xs.reshape(-1,1))
    plt.plot(xs,ys,linewidth=1,label=act)

plt.legend(loc="lower left")
plt.ylim(-1,1)
plt.xlabel("x")
plt.ylabel("y")
plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
y
0.0
0.2
0.4
0.6
0.8
1.0
x
Data
identity
logistic
tanh
relu

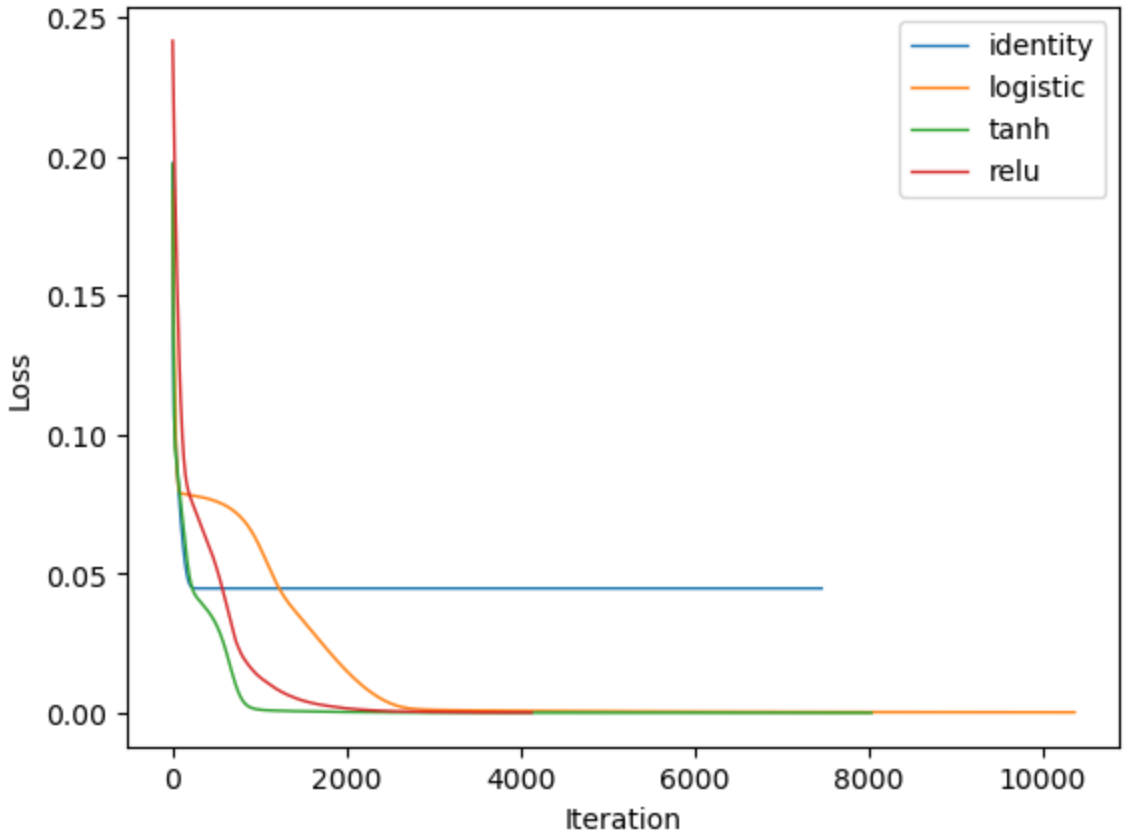
Loss curves

Now, create another loss curve plot, but this time, include all four MLP models with a legend indicating which activation function corresponds to each curve.

In [5]:
```
# YOUR CODE GOES HERE

for act in activations:
    model = models[act]
    loss = model.loss_curve_
    plt.plot(loss,linewidth = 1,label = act)

plt.xlabel("Iteration")
plt.ylabel("Loss")
plt.legend()
plt.show()
```



Questions

1. Which activation functions produced a good fit?

   relu, tanh, logistic activation functions produced a good fit.

2. Which activation function's model converged the "slowest"?

   Logistic activation function's model converged the slowest.

3. Of the networks that fit well, which activation function's model converged the "fastest"?

   tanh activation function's model converged the fastest.

In [ ]: